Neural Network Training with Approximate Logarithmic Computations

International Conference on Acoustics, Speech and Signal Processing

Arnab Sanyal

Ming Hsieh Department of Electrical & Computer Engineering
University of Southern California
Los Angeles, CA, USA

Virtual Tele-conference Presentation, May 2020

Goals

Neural Network Training with Approximate Logarithmic Computations

- Enabling Neural Network training on edge-devices.

Goals

- Enabling Neural Network training on edge-devices.
- Computation reduction is of paramount importance.

Goals

Neural Network Training with Approximate Logarithmic Computations

- Enabling Neural Network training on edge-devices.
- Computation reduction is of paramount importance.
- Variety of approaches already exist - sparsity, pruning, quantization.

Goals

- Enabling Neural Network training on edge-devices.
- Computation reduction is of paramount importance.
- Variety of approaches already exist - sparsity, pruning, quantization.
- Our method – design end-to-end training in a logarithmic number system (LNS).

Goals

- Enabling Neural Network training on edge-devices.
- Computation reduction is of paramount importance.
- Variety of approaches already exist - sparsity, pruning, quantization.
- Our method – design end-to-end training in a logarithmic number system (LNS).
    - All NN operations needs to be defined in LNS

Goals

Neural Network Training with Approximate Logarithmic Computations

- Enabling Neural Network training on edge-devices.
- Computation reduction is of paramount importance.
- Variety of approaches already exist - sparsity, pruning, quantization.
- Our method – design end-to-end training in a logarithmic number system (LNS).
  - All NN operations needs to be defined in LNS
  - In LNS multiplications are cheap but addition are computationally expensive

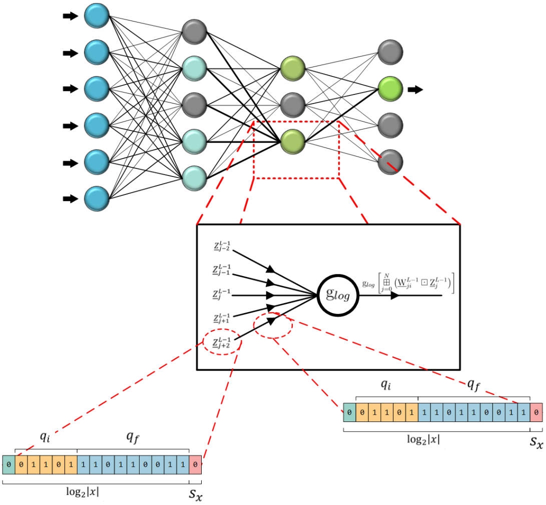
Goals

Neural Network Training with Approximate Logarithmic Computations

- Enabling Neural Network training on edge-devices.
- Computation reduction is of paramount importance.
- Variety of approaches already exist - sparsity, pruning, quantization.
- Our method – design end-to-end training in a logarithmic number system (LNS).
  - All NN operations needs to be defined in LNS
  - In LNS multiplications are cheap but addition are computationally expensive
  - Resort to **Approximate Logarithmic Fixed-Point Computations**

LNS Neural Network Pipeline

Neural Network Training with Approximate Logarithmic Computations



ICASSP2020 Barcelona

UNIVERSITY OF SOUTHERN CALIFORNIA 1880

Equivalence

$$v \longleftrightarrow \underline{V} = (V, s_v)$$
$$V = \log_2(|v|)$$
$$s_v = \mathrm{sign}(v)$$

Equivalence

$$v \longleftrightarrow \underline{V} = (V, s_v)$$
$$V = \log_2(|v|)$$
$$s_v = \mathrm{sign}(v)$$

Multiplication

$$u = xy \longleftrightarrow \underline{U} = \underline{X} \boxdot \underline{Y}$$
$$U = X + Y$$
$$s_u = \overline{(s_x \veebar s_y)}$$

Addition

$$z = x + y \longleftrightarrow \underline{Z} = \underline{X} \boxplus \underline{Y}$$

$$Z = \begin{cases} \max(X, Y) + \Delta_{+}(|X - Y|) & s_x = s_y \\ \max(X, Y) + \Delta_{-}(|X - Y|) & s_x \neq s_y \end{cases}$$

$$s_z = \begin{cases} s_x & X > Y \\ s_y & X \leq Y \end{cases}$$

Mathematical Operations

Neural Network Training with Approximate Logarithmic Computations

Addition

$$z = x + y \longleftrightarrow \underline{Z} = \underline{X} \boxplus \underline{Y}$$

$$Z = \begin{cases} \max(X, Y) + \Delta_+ \left(|X - Y|\right) & s_x = s_y \\ \max(X, Y) + \Delta_- \left(|X - Y|\right) & s_x \neq s_y \end{cases}$$

$$\Delta_+(d) = \log_2 \left(1 + 2^{-d}\right) \qquad d \geq 0$$

$$\Delta_-(d) = \log_2 \left(1 - 2^{-d}\right) \qquad d \geq 0$$

$$s_z = \begin{cases} s_x & X > Y \\ s_y & X \leq Y \end{cases}$$

Subtraction

$$t = x - y \longleftrightarrow \underline{T} = \underline{X} \boxminus \underline{Y} = \underline{X} \boxplus (Y, \overline{s_y})$$

Subtraction	$t = x - y \longleftrightarrow \underline{T} = \underline{X} \boxminus \underline{Y} = \underline{X} \boxplus (Y, \overline{s_y})$
Exponentiation	$w = x^y \longleftrightarrow \underline{W} = (yX, 1)$

Subtraction

$$t = x - y \longleftrightarrow \underline{T} = \underline{X} \boxminus \underline{Y} = \underline{X} \boxplus (Y, \overline{s_y})$$

Exponentiation

$$w = x^y \longleftrightarrow \underline{W} = (yX, 1)$$

Approximate additions

Mathematical Operations

Neural Network Training with Approximate Logarithmic Computations

Subtraction	$t = x - y \longleftrightarrow \underline{T} = \underline{X} \boxminus \underline{Y} = \underline{X} \boxplus (Y, \overline{s_y})$
Exponentiation	$w = x^y \longleftrightarrow \underline{W} = (yX, 1)$

Approximate additions

- Approximate Δ to reduce addition computation complexity

Subtraction	$t = x - y \longleftrightarrow \underline{T} = \underline{X} \boxminus \underline{Y} = \underline{X} \boxplus (Y, \overline{s_y})$
Exponentiation	$w = x^y \longleftrightarrow \underline{W} = (yX, 1)$

Approximate additions

- Approximate Δ to reduce addition computation complexity
- Two different approximations explored

Mathematical Operations

Neural Network Training with Approximate Logarithmic Computations

Subtraction	$t = x - y \longleftrightarrow \underline{T} = \underline{X} \boxminus \underline{Y} = \underline{X} \boxplus (Y, \overline{s_y})$
Exponentiation	$w = x^y \longleftrightarrow \underline{W} = (yX, 1)$

Approximate additions

- Approximate Δ to reduce addition computation complexity
- Two different approximations explored
  - Look-Up Table (LUT) based approximations

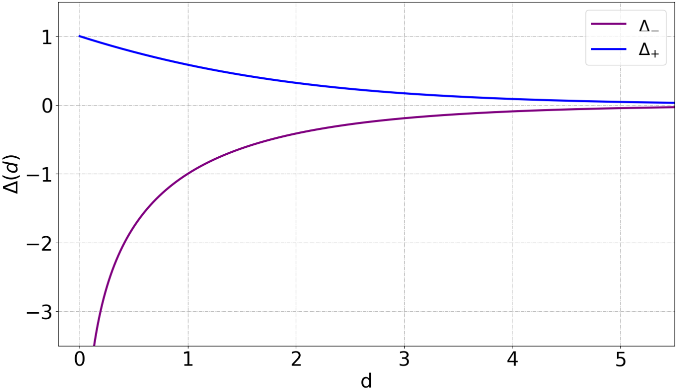
Subtraction	$t = x - y \longleftrightarrow \underline{T} = \underline{X} \boxminus \underline{Y} = \underline{X} \boxplus (Y, \overline{s_y})$
Exponentiation	$w = x^y \longleftrightarrow \underline{W} = (yX, 1)$

Approximate additions

- Approximate Δ to reduce addition computation complexity
- Two different approximations explored
  - Look-Up Table (LUT) based approximations
  - Bit-shift (BS) based approximations

Visualizing Δ from LNS $\boxplus$ Additions

Neural Network Training with Approximate Logarithmic Computations



Look-Up Table Based Approximations

- Idea - Store the Δ terms as a Look-Up Table (LUT)

Look-Up Table Based Approximations

Neural Network Training with Approximate Logarithmic Computations

- Idea - Store the Δ terms as a Look-Up Table (LUT)
- LUT specified by three parameters – fixed point width, dynamic range, resolution

Look-Up Table Based Approximations

Neural Network Training with Approximate Logarithmic Computations

- Idea - Store the Δ terms as a Look-Up Table (LUT)
- LUT specified by three parameters – fixed point width, dynamic range, resolution
  - Dynamic range determined by fixed point width as Δ terms die down to 0

Look-Up Table Based Approximations

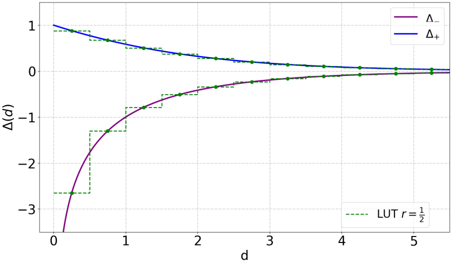
Neural Network Training with Approximate Logarithmic Computations

- Idea - Store the Δ terms as a Look-Up Table (LUT)
- LUT specified by three parameters – fixed point width, dynamic range, resolution
  - Dynamic range determined by fixed point width as Δ terms die down to 0
  - Resolution is a hyper-parameter

- Idea - Store the Δ terms as a Look-Up Table (LUT)
- LUT specified by three parameters – fixed point width, dynamic range, resolution
    - Dynamic range determined by fixed point width as Δ terms die down to 0
    - Resolution is a hyper-parameter
    - Fixed-point width is a hyper-parameter

Look-Up Table Based Approximations

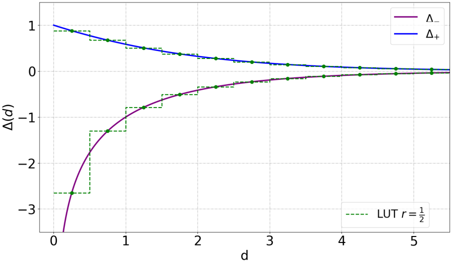
Neural Network Training with Approximate Logarithmic Computations

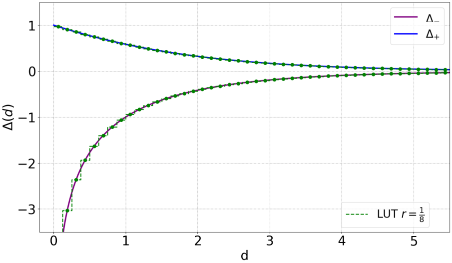


LUT Resolution $r = \frac{1}{2}$

Look-Up Table Based Approximations

Neural Network Training with Approximate Logarithmic Computations





LUT Resolution $r = \frac{1}{2}$

LUT Resolution $r = \frac{1}{8}$

Taylor Series approximation

$$\log_e(1 \pm x) \approx \pm x \qquad 0 \leq x \ll 1$$

$$\begin{aligned}\Delta_{\pm}(d) &= \log_2\left(1 \pm 2^{-d}\right) \\ &\approx \pm \log_2 e \times 2^{-d}\end{aligned}$$

Bit-Shift Based Approximations

Neural Network Training with Approximate Logarithmic Computations

Taylor Series approximation

$$\log_e(1 \pm x) \approx \pm x \qquad 0 \leq x \ll 1$$

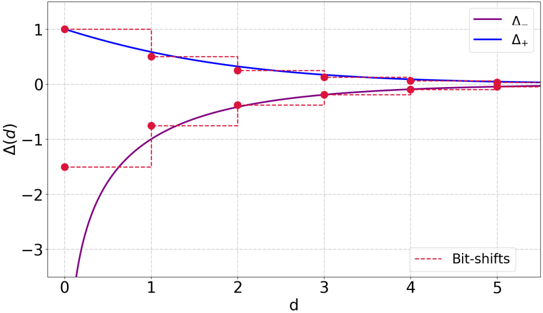
$$\begin{aligned}\Delta_{\pm}(d) &= \log_2\left(1 \pm 2^{-d}\right)\\ &\approx \pm \log_2 e \times 2^{-d}\end{aligned}$$

Further Simplification

$$\begin{aligned}\log_2 e = 1.442695\cdots &\approx 1.4375 = 2^0 + 2^{-1} - 2^{-4}\\ &\approx 1.5 = 2^0 + 2^{-1}\\ &\approx 1 = 2^0\end{aligned}$$

Bit-Shift Based Approximations

Neural Network Training with Approximate Logarithmic Computations



$$\Delta_+(d) \approx \mathbb{BS}(1, -d)$$
$$\Delta_-(d) \approx -\mathbb{BS}(1.5, -d)$$

Can be thought of as a LUT with $r = 1$

Multiply accumulate
(MAC)

The LNS Neuron

Neural Network Training with Approximate Logarithmic Computations

Multiply accumulate (MAC)

$$z_i = \sum_j w_{ij} x_j + b_i \longleftrightarrow \underline{Z}_i = \boxplus_j \underline{W}_{ij} \boxdot \underline{X}_j \boxplus \underline{B}_i$$

The LNS Neuron

Neural Network Training with Approximate Logarithmic Computations

Multiply accumulate (MAC)

$$z_i = \sum_j w_{ij} x_j + b_i \longleftrightarrow \underline{Z}_i = \boxplus_j \underline{W}_{ij} \boxdot \underline{X}_j \boxplus \underline{B}_i$$

LNS bit-width Constraint

The LNS Neuron

Neural Network Training with Approximate Logarithmic Computations

Multiply accumulate (MAC)

$$z_i = \sum_j w_{ij} x_j + b_i \longleftrightarrow \underline{Z}_i = \boxplus_j \underline{W}_{ij} \boxdot \underline{X}_j \boxplus \underline{B}_i$$

LNS bit-width Constraint

$$W_{\log} \geq 1 + \max\left(\lceil \log_2 \left(b_i + 1\right) \rceil, \lceil \log_2 b_f \rceil\right) + W_{\mathrm{lin}}$$

Experiments suggest that $W_{\log} \approx W_{\mathrm{lin}}$ suffices in practice. In this work, set $W_{\log} = W_{\mathrm{lin}}$

LNS Weight Initialization

- To avoid hundreds of thousands of parameter initialization on prior distribution and taking logarithm of them, use standard change of measure approaches from probability to derive desired distribution

LNS Weight Initialization

Neural Network Training with Approximate Logarithmic Computations

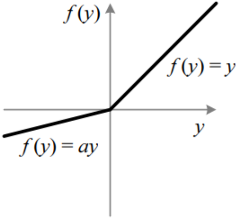
- To avoid hundreds of thousands of parameter initialization on prior distribution and taking logarithm of them, use standard change of measure approaches from probability to derive desired distribution
- Weights generally initialized from symmetric distributions. Hence the sign parameter can be initialized randomly and independently of the magnitude from a $Bernoulli(\frac{1}{2})$ distribution.

- To avoid hundreds of thousands of parameter initialization on prior distribution and taking logarithm of them, use standard change of measure approaches from probability to derive desired distribution
- Weights generally initialized from symmetric distributions. Hence the sign parameter can be initialized randomly and independently of the magnitude from a *Bernoulli*$(\frac{1}{2})$ distribution.
- The magnitude distribution for weights reduce to,

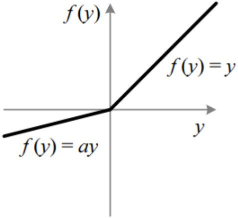
$$f_W(y) = 2^{y+1} \times \log_e 2 \times f_w(2^y)$$

The Log-Leaky ReLU Activation

Neural Network Training with Approximate Logarithmic Computations



Parametric ReLU

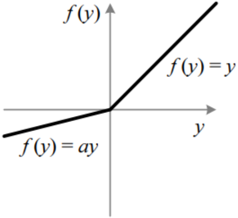


Parametric ReLU

- First proposed in ICCV 2015, fixes the *Dying ReLU* problem

The Log-Leaky ReLU Activation

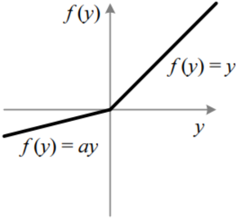
Neural Network Training with Approximate Logarithmic Computations



Parametric ReLU

- First proposed in ICCV 2015, fixes the *Dying ReLU* problem
- Attractive to this research as Dying ReLU could make activations ∞ in LNS

The Log-Leaky ReLU Activation



Parametric ReLU

- First proposed in ICCV 2015, fixes the *Dying ReLU* problem
- Attractive to this research as Dying ReLU could make activations ∞ in LNS
- This brings us to *Log-Leaky ReLU*

$$g_{\text{llReLU}}\left((X, s_x) \,\middle|\, \beta\right) = \begin{cases} (X, s_x) & s_x = 1 \\ (X + \beta, s_x) & s_x = 0 \end{cases}$$

Gradient calculation

Gradient calculation

$$p_{ij} = \frac{e^{a_{ij}}}{\sum_{j=1}^{N} e^{a_{ij}}}$$

$$\delta_{ij} = p_{ij} - y_{ij}$$

LNS Soft-Max

Neural Network Training with Approximate Logarithmic Computations

Gradient calculation

$$p_{ij} = \frac{e^{a_{ij}}}{\sum_{j=1}^{N} e^{a_{ij}}}$$

$$\delta_{ij} = p_{ij} - y_{ij}$$

Log-probabilities

LNS Soft-Max

Neural Network Training with Approximate Logarithmic Computations

Gradient calculation

$$p_{ij} = \frac{e^{a_{ij}}}{\sum_{j=1}^{N} e^{a_{ij}}}$$

$$\delta_{ij} = p_{ij} - y_{ij}$$

Log-probabilities

$$\log_2 p_{ij} = (a_{ij} \log_2 e) - \boxplus_{j=1}^{N} (a_{ij} \log_2 e, 1)$$

LNS Soft-Max

Neural Network Training with Approximate Logarithmic Computations

Gradient calculation

$$p_{ij} = \frac{e^{a_{ij}}}{\sum_{j=1}^{N} e^{a_{ij}}}$$

$$\delta_{ij} = p_{ij} - y_{ij}$$

Log-probabilities

$$\log_2 p_{ij} = (a_{ij} \log_2 e) - \boxplus_{j=1}^{N} (a_{ij} \log_2 e, 1)$$

LNS gradients

LNS Soft-Max

Neural Network Training with Approximate Logarithmic Computations

Gradient calculation

$$p_{ij} = \frac{e^{a_{ij}}}{\sum_{j=1}^{N} e^{a_{ij}}}$$

$$\delta_{ij} = p_{ij} - y_{ij}$$

Log-probabilities

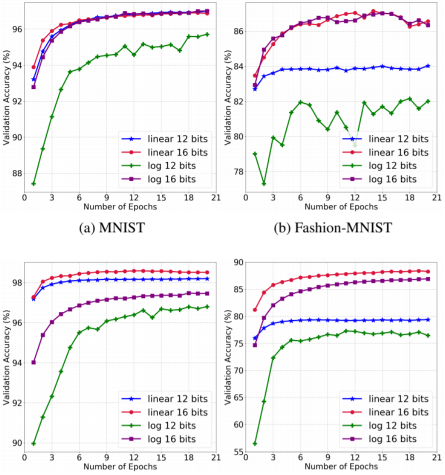
$$\log_2 p_{ij} = (a_{ij} \log_2 e) - \underset{j=1}{\overset{N}{\boxplus}} (a_{ij} \log_2 e, 1)$$

LNS gradients

$$\left(\log_2 |\delta_{ij}|, s_{\delta_{ij}}\right) = \underline{P}_{ij} \boxminus \left(\log_2 |y_{ij}|, s_{y_{ij}}\right)$$

Numerical Results

Neural Network Training with Approximate Logarithmic Computations



(a) MNIST

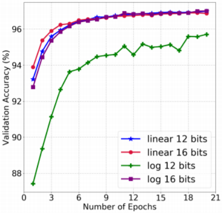
(b) Fashion-MNIST

(c) EMNIST-digits

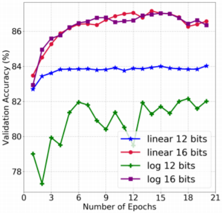
(d) EMNIST-letters

Numerical Results

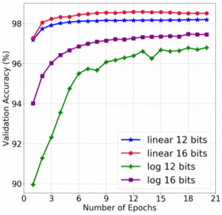
Neural Network Training with Approximate Logarithmic Computations



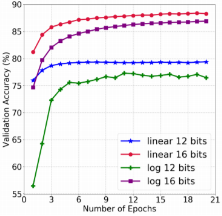
(a) MNIST



(b) Fashion-MNIST



(c) EMNIST-digits



(d) EMNIST-letters

Datasets	Float	**Linear-domain** fixed-point		**Log-domain** fixed-point look-up tables		**Log-domain** fixed-point bit-shifts	
		12b	16b	12b	16b	12b	16b
MNIST	97.4	97.3	96.9	96.0	97.2	95.5	96.5
FMNIST	87.1	82.8	88.0	80.5	87.1	79.3	85.7
EMNISTD	98.6	98.3	98.7	96.9	97.5	96.2	97.4
EMNISTL	88.1	79.7	88.7	76.4	86.7	73.7	82.5

Number of epochs trained $= 20$
Size of tables $= 20(r = \frac{1}{2})$; soft-max uses 640 element tables$(r = \frac{1}{64})$
Table shows test-set accuracy

Future Works

Neural Network Training with Approximate Logarithmic Computations

- Extend Future work to CNNs on harder datasets

Future Works

- Extend Future work to CNNs on harder datasets
- Better Approximation Design

Future Works

- Extend Future work to CNNs on harder datasets
- Better Approximation Design
    - Functional Approximations using constrained optimization

Future Works

- Extend Future work to CNNs on harder datasets
- Better Approximation Design
  - Functional Approximations using constrained optimization
  - Replacing Soft-max Layer with multi-class Sigmoid

Future Works

- Extend Future work to CNNs on harder datasets
- Better Approximation Design
  - Functional Approximations using constrained optimization
  - Replacing Soft-max Layer with multi-class Sigmoid
- Reliability and Robustness Analysis

Future Works

- Extend Future work to CNNs on harder datasets
- Better Approximation Design
  - Functional Approximations using constrained optimization
  - Replacing Soft-max Layer with multi-class Sigmoid
- Reliability and Robustness Analysis
  - Cost-Accuracy trade-off across different approximations

Future Works

- Extend Future work to CNNs on harder datasets
- Better Approximation Design
  - Functional Approximations using constrained optimization
  - Replacing Soft-max Layer with multi-class Sigmoid
- Reliability and Robustness Analysis
  - Cost-Accuracy trade-off across different approximations
  - Weight-Activation Relation mapping for LNS-neurons using Supervised Learning (LDA, QDA)

Acknowledgements

This work is supported in part by the National Science Foundation (CCF-1763747)